PRICE $ 2.00

THE 80 NOTEBOOK

JUNE 1980 ISSUE #3

NOTE: The term "TRS-80" is a registered trademark of Radio Shack, a Division of Tandy Corporation. THE 80 NOTEBOOK is not affiliated with Radio Shack or Tandy Corporation in any way.

LETTERS TO THE EDITOR

Michael,

Regarding the dice roll random number generator in the Monopoly game printed in the April 1980 issue of THE 80 NOTEBOOK.

I disagree with lines 60, 70, 3840 and 3850 as being random number generators. All four will be weighted heavily on ones. In fact, there will be nearly twice as many ones as there are any other number generated. IF X=Ø THEN X=1 causes this. Furthermore, the random numbers generated will be no bigger than 5. RND(Ø) generates numbers between Ø and 1 not inclusive of Ø and 1. The largest non-integer generated in single precision is .999999 and multiplying this by 6 nets a 5.99994 and making an integer out of that gives you a maximum of low and behold 5.

There are a couple ways to alleviate the problem. One is to remove the statement IF X=Ø THEN X=1 and replace it with X=X+1 or just strike all four lines and use the statement X=RND(6).

Also, line 3290 AJ=-15Ø should be AJ=15Ø and 347Ø AG=-4ØØ should be AG=4ØØ.

Sincerely,

Don McDougall, W6OA
744 Camelia Drive
Livermore, CA 94550

Dear Sir:

I have just today received my second copy of THE 80 NOTEBOOK. I am pleased so far. I am writing about the MONOPOLY program which appeared in your first issue.

When I copy a program, I also expect a number of errors caused by my typing. This program was no exception. There were some mistakes that I seem to have found and have not been able to find. I noticed that a "6" never appeared on the roll of the dice. Should lines # 60 & # 70 be corrected to read:

```
60 X=INT(RND(0*7):IFX=0THENX=1

70 Y=INT(RND(0*7):IFY=0THENY=1
```

I also ran across a difficulty, only once with line # 7260, which said that the NEXT3 should be before the NEXT4. Should it? I also have had difficulties with the COMMUNITY CHEST. It keeps asking "Do you wish to purchase it". And a few other small bugs. Probably my typing and they have not as yet been found.

What I would really like to know, "Has anyone expanded this program to list the properties held by the different players, especially according to color." Has anyone written a subroutine for the sale of properties amongst the players? This would be a tremendous addition. I have 32K of memory so this is no problem for me. If they have, I would be interested in the expanded program.

Keep up the good work.

A fellow Keystone Computer Addict,

Fr. William M. Kuba
St. Bibiana's Parish
111 Germania St.
Galeton, PA 16922

NEW PRODUCTS

TAKE NOTE! THE MUSIC BOX IS HERE!

Newtech Computer Systems, a leading manufacturer of music peripherals and software for S-100 and SS-50 computers, introduces The Music Box. The Music Box is a complete hardware/software tool that enables you to produce music and sound effects on your TRS-80.

You can compose music, play or sing along with the computer, or just listen to your favorite tunes — up to four notes at a time, with a seven octave range. And you can make it sound like one, two, three or four different musical instruments at the same time. Or you can make all sorts of sound effects and noises like explosions, gun shots, "phasor" and other space war sounds.

The Music Box plugs directly into the TRS-80 keyboard or the Expansion Interface

Bus Extension. It includes a volume control, a 400 milliwatt power amp, and phono jack for easy connection to an external speaker. Software is supplied on Level II cassette. Requires a 32K Ram or larger Level II computer.

Future software developments include more pre-coded music, games with sound effects, music education, telephone tone dialing, and Morse code programs.

$249 complete with software and users manual. Add $3 for shipping and $1 if COD.

NEWTECH COMPUTER SYSTEMS, INC.
230 Clinton Street
Brooklyn, New York 11201
(212) 625-6220

EMTROL INTRODUCES "LYNX" DIRECT-CONNECT MODEM FOR TRS-80

Lancaster, Pennsylvania — Emtrol Systems, Inc., 1262 Loop Road, has introduced LYNX, a new direct-connect telephone modem designed for the Radio Shack TRS-80 microcomputer.

LYNX comprises a total telephone linkage system in one package, eliminating the need for a separate expansion interface, interface board, telephone coupler and communications software. It is priced at $239.95 (less tax), including "terminal" program on cassette, instruction manual and power pack.

LYNX connects directly with the TRS-80 keyboard and the telephone line; no acoustic coupler is used. It includes originate and answer capability, and is programmable for word length, parity, number of stop bits and full or half duplex.

Minumum hardware requirements for LYNX are a TRS-80 Level I or II with 4K RAM. Pending FCC registration permits direct plug-in connection with the telephone line. During data exchange, LYNX automatically disconnects the local telephone handset, thus eliminating room noise pickup typical of acoustic couplers.

The LYNX instruction manual describes time-share access methods, such as "The Source," CBBS, Forum-80 and TRS-80-to-TRS-80 links.

LYNX will be available through dealers or direct from the factory after June 1, 1980. VISA and Master Card will be accepted on factory orders.

LETTER TO THE EDITOR

Dear Editor:

Not very long ago, I was called by a friend who was having all kinds of trouble making his serial Diablo printer work with Scripsit. I sat down with him and he explained the horror story he lived through trying to get an explanation out of the folks at Radio Shack about how to get their serial driver to work. I worked with him

and came up with the enclosed assembly language RS-232 Driver for Scripsit.

As it says in the source listing, this program is an interfacing of the Decwriter serial driver described in the Radio Shack RS-232 Manual and the Serial Driver described in the Scripsit Manual. I think this program would be of great value to your readership.

I hope you will consider it for publication.

If you have any questions or comments, you can get in touch with me through the Kansas City FORUM 80 system (816) 861-7040, the SOURCE MAIL (TCF035) or my home (816) 756-1847.

Sincerely,

Jim Cambron
J. A. Cambron Co.
P.O. Box 10005
Kansas City, MO 64111

SCRIPSIT SERIAL PRINTER DRIVER LISTING

```
100 ;       SCRIPSIT SERIAL PRINTER DRIVER.
110 ;       ADAPTED FROM RADIO SHACK DOCUMENTATION
120 ;       (SCRIPSIT MANUAL AND RS-232 MANUAL)
130 ;       BY JIM CAMBRON P.O. BOX 10005 K.C. MO 64111
140 ;
150 ;   THIS ASSEMBLY LANGUAGE ROUTINE LOADS SCRIPSIT FROM DISK,
160 ; INITIALIZES A SERIAL PRINTER ROUTINE, PATCHES THE SERIALROUTINE
170 ; INTO SCRIPSIT, AND JUMPS TO THE ENTRY LOCATION OF SCRIPSIT.
180 ; THIS PROGRAM IS WRITTEN FOR DISK OPERATION ONLY! LOAD IT ON
190 ; DISK USING TAPEDISK OR LMOFFSET. WHEN YOU WANT TO RUN YOUR
200 ; SERIAL PRINTER WITH SCRIPSIT, ENTER THE FILENAME OF THIS
210 ; PROGRAM INSTEAD OF SCRIPSIT.
220 ;   THE SERIAL DRIVER USED IN THIS PROGRAM IS THE DECWRITER
230 ; DRIVER FROM THE RADIO SHACK RS-232 MANUAL. THIS DRIVER WAS
240 ; REWRITTEN TO WORK WITH THIS PROGRAM AS FOLLOWS:
250 ;
260 ; 1)  THE INITIALIZATION PART OF THE DECWRITER DRIVER WAS PLACED
270 ; IN THE LOCATION INDICATED BY THE SCRIPSIT DOCUMENTATION.
280 ; THE EQUATES WERE MOVED TO THE BEGINNING OF THE SOURCE.
290 ;
300 ; 2)  AS MENTIONED IN THE SCRIPSIT DOCUMENTATION, THE CHARACTER
310 ; TO BE PRINTED APPEARS IN THE 'A' REGISTER. THE DECWRITER
320 ; DRIVER PRINTS THE CONTENTS OF THE 'C' REGISTER. TO FIX THIS,
330 ; LINE 820 WAS ADDED TO THE SOURCE CODE TO MOVE THE CONTENTS
340 ; OF THE 'A' REGISTER INTO THE 'C' REGISTER SO THE DECWRITER
350 ; DRIVER COULD PRINT THE CORRECT CHARACTER.
360 ;
370 ; THIS SOURCE CODE GENERATES OBJECT FOR A 32K SYSTEM. YOU
380 ; CAN CHANGE THIS TO 48K BY CHANGING LINE 520 TO READ:
390 ;
```

```
                400 ;       Ø52Ø       ORG       ØFFØØH
                410 ;
                420 ; BEFORE ASSEMBLING THIS TO AN OBJECT FILE.
                430 ;
                440 ;    THIS DRIVER HAS SUCCESSFULLY DRIVEN A DIABLO PRINTER
                450 ; ON A 32K SYSTEM. DON'T FORGET TO SET THE SWITCH INSIDE
                460 ; THE EXPANSION INTERFACE ON THE RS-232 CARD TO 'COMM'!
                470 ;
ØØE8            480 RESURT  EQU     ØE8H
ØØE9            490 SWITCH  EQU     ØE9H
ØØEA            500 CNTREG  EQU     ØEAH
ØØEB            510 DTAREG  EQU     ØEBH
BFØØ            520         ORG     ØBFØØH
BFØØ            530 SERIAL  EQU     $
BFØØ E5         540         PUSH    HL
BFØ1 218CBF     550         LD      HL,ELBOW
BFØ4 34         560         INC     (HL)
BFØ5 35         570         DEC     (HL)
BFØ6 E1         580         POP     HL
BFØ7 2Ø3Ø       590         JR      NZ,DRIVER
BFØ9 118DBF     600         LD      DE,DCBADR
BFØC CD3Ø44     610         CALL    LOAD
BFØF CD2844     620         CALL    CLOSE
BF12 3EØ1       0630         LD      A,1
BF14 328CBF     0640         LD      (ELBOW),A
BF17 3E21       0650         LD      A,21H
BF19 326752     0660         LD      (5267H),A
BF1C 21FFBE     0670         LD      HL,SERIAL-1
BF1F 226852     0680         LD      (5268H),HL
BF22 3EC3       0690         LD      A,ØC3H
BF24 323966     0700         LD      (6639H),A
BF27 32565F     0710         LD      (5F56H),A
BF2A 2159BF     0720         LD      HL,INIT
BF2D 223A66     0730         LD      (663AH),HL
BF3Ø 21ØØBF     0740         LD      HL,SERIAL
BF33 22575F     0750         LD      (5F57H),HL
BF36 C3ØØ52     0760         JP      52ØØH          ; GOTO SCRIPSIT.
BF39            0770 DRIVER EQU     $
BF39 Ø8         0780         EX      AF,AF'         ; SWITCH AF REG
                0790 ;       THIS IS THE DRIVER PART OF THE DECWRITER
                0800 ;       ROUTINE FOUND IN THE RS-232 MANUAL.
                0810 ;
BF3A 4F         0820         LD      C,A            ; THE BIG FIX!!!!
BF3B DBEA       0830 STATIN  IN      A,(CNTREG)
BF3D CB77       0840         BIT     6,A
BF3F 28FA       0850         JR      Z,STATIN
BF41 79         0860         LD      A,C
BF42 D3EB       0870         OUT     (DTAREG),A
BF44 FEØD       0880         CP      ØDH
BF46 2ØØ4       0890         JR      NZ,RETRN
BF48 ØEØA       0900         LD      C,ØAH
BF4A 18EF       0910         JR      STATIN
BF4C C3745F     0920 RETRN   JP      5F74H          ; BLOW THIS POPSTAND
BF4F 22         0930 BDTABL  DEFB    22H
```

```
BF5Ø 44      0940           DEFB    44H
BF51 55      0950           DEFB    55H
BF52 66      0960           DEFB    66H
BF53 77      0970           DEFB    77H
BF54 AA      0980           DEFB    ØAAH
BF55 CC      0990           DEFB    ØCCH
BF56 EE      1000           DEFB    ØEEH
BF57 ØØ      1010 SWTIMG    DEFB    ØØH
BF58 ØØ      1020 FLAG      DEFB    ØØH
             1030 ;
BF59         1040 INIT      EQU     $
BF59 3EØ1    1050           LD      A,1
BF5B 32627C  1060           LD      (7C62H),A
BF5E E5      1070           PUSH    HL
BF5F C5      1080           PUSH    BC
BF6Ø F5      1090           PUSH    AF
BF61 3A58BF  1100           LD      A,(FLAG)
BF64 FEØ1    1110           CP      Ø1H
BF66 2820    1120           JR      Z,RESTOR
BF68 3EØ1    1130           LD      A,Ø1H
BF6A 3258BF  1140           LD      (FLAG),A
BF6D D3E8    1150           OUT     (RESURT),A
BF6F DBE9    01160           IN      A,(SWITCH)
BF71 E6F8    01170           AND     ØF8H
BF73 F6Ø4    01180           OR      Ø4H
BF75 3257BF  01190           LD      (SWTIMG),A
BF78 D3EA    01200           OUT     (CNTREG),A
BF7A DBE9    01210           IN      A,(SWITCH)
BF7C E607    01220           AND     Ø7H
BF7E 214FBF  01230           LD      HL,BDTABL
BF81 Ø6ØØ    01240           LD      B,ØØH
BF83 4F      01250           LD      C,A
BF84 Ø9      01260           ADD     HL,BC
BF85 7E      01270           LD      A,(HL)
BF86 D3E9    01280           OUT     (SWITCH),A
BF88 F1      01290 RESTOR    POP     AF
BF89 C1      01300           POP     BC
BF8A E1      01310           POP     HL
BF8B C9      01320           RET                          ; WEEZE DONE HERE
             01330 ;
BF8C ØØ      01340 ELBOW     DEFB    ØØH
BF8D 53      01350 DCBADR    DEFM    'SCRIPSIT/LC'        ; SCRIPSIT FILENAME
BF98 Ø3      01360           DEFB    3
443Ø         01370 LOAD      EQU     443ØH
4428         01380 CLOSE     EQU     4428H
BFØØ         01390           END     SERIAL
ØØØØØ TOTAL ERRORS


BDTABL BF4F 00930    01230
CLOSE  4428 01380    00620
CNTREG ØØEA 00500    00830 01200
DCBADR BF8D 01350    00600
DRIVER BF39 00770    00590
```

DTAREG	ØØEB	00510	00870
ELBOW	BF8C	01340	00550 00640
FLAG	BF58	01020	01100 01140
INIT	BF59	01040	00720
LOAD	443Ø	01370	00610
RESTOR	BF88	01290	01120
RESURT	ØØE8	00480	01150
RETRN	BF4C	00920	00890
SERIAL	BFØØ	00530	00670 00740 01390
STATIN	BF3B	00830	00850 00910
SWITCH	ØØE9	00490	01160 01210 01280
SWTIMG	BF57	01010	01190

.....

NEW PRODUCTS

.....

ZOOM3.6

ZOOM3.6 is an electronic "black-box" that connects in between a Level-II TRS-80 (or Expansion-Interface) and a CTR-41 or CTR-80 cassette recorder. With the ready-to-run software supplied with it, tapes can be written and read in a special format at 3600 bits per second - over 7 times faster than Radio Shack's 500! At that speed, 2K of RAM loads in under 5 seconds, and 16K loads in just 36 seconds - instead of in almost four-and-a-half minutes.

No soldering and no modifications are needed. ZOOM3.6 is wholly transparent to all the XRX mods (and to any other signal processor connected to the cassette-port), and to the CLOAD, CSAVE, SYSTEM, and PUNCH functions. A built-in relay and a toggle-switch on the front panel bypass the unreliable Radio Shack reed relay. All cables can be left plugged in permanently.

ZOOM3.6 works at any volume setting between typically 2 and 8 on a CTR-41 (the range is a little narrower on a CTR-80).

The menu of the accompanying stand-alone monitor (ZMBUG V1.00) supplies the following options:

(1) Input or Output machine code (equivalent to the SYSTEM and TBUG PUNCH facilities, but with 8-character filenames).

(2) Load or Save a BASIC program (equivalent to the CLOAD and CSAVE functions, but with 8-character filenames).

(3) Read or Write a test tape for instant visual feedback as to whether or not your system is correctly set up.

(4) Jump back to the BASIC monitor.

(5) Execute machine code that has just been loaded.

The Out and Save routines can be commanded to write the code any number of times (up to 255) in a row, without supervision.

Certified cassettes are not essential. Obviously the system requires tape without audible dropouts, but otherwise it does not seem too fussy. A user can expect minimal maintenance to give an error rate so low that, for example, 42.2-kilobyte files can be read repeatedly without error.

The computer will not crash if it finds an error, or even if it is made to start or stop reading in the middle of a recording, and the system will not hang-up while searching for a file. BREAK causes a return to the menu during reads and writes.

For a small charge, a user can purchase the commented source-code for ZMBUG, so that it may be copied from, adapted, or relocated. Written instructions plus a sample of assembly-language source code necessary to patch ZOOM3.6 drivers into THE ELECTRIC PENCIL, are available also. (The patches may need modification by the user, if he owns a different version of THE ELECTRIC PENCIL, however.) Patches for EDTASM and other software will be available later.

Since all timing is done in hardware, the read/write routines are short (about 150 bytes), and there is over 200 microseconds between bits in which to calculate checksums or to do whatever is needed. This makes it relatively easy for a user to patch ZOOM3.6 read/write drivers into his own software.

ZOOM3.6 is the fastest CTR-41/CTR-80-based system available, and it is believed truly to represent the state-of-the-art. With ZMBUG V1.0 object code and the manual, it costs only $119.00; far less than either an Expansion-Interface plus a floppy-disk ($670 and up) or a high-speed cassette deck ($250 and up). Delivery is currently estimated at three weeks.

For further information, please send a SASE to ZOOM! PO Box 3766, Nashua, NH 03061. (tel. 603-889-0901)

••

BASIC PROGRAMMING AND THE ART OF SIMULATION

In an earlier article, we took a look at the programming instruction set available to you in Level I BASIC and what each instruction is capable of.

In this article, we will discuss how to best combine those instructions into a meaningful program to accomplish a series of tasks. We will also take a look at those Level II BASIC instructions that can best serve us in programs designed to simulate an activity or series of activities and how they work.

You might ask, "What does simulation have to do with programming?". If you study the true function of programming, you will find that every program ever written is a computerized model of some activity or thought process. This computer model can be considered a simulation of the activity or thought process it represents.

To spite this fact, the term "computer simulation" most often refers to a program written to simulate the processes involved in the operation of a machine, factory or organization.

This kind of simulation often must react to input data in what is called "real-time". This means that the program must act like the entity it represents when it encounters a stimulus the real entity would receive.

The output of such a simulation is often a timetable showing what the stimulus was, how it reacted to it, how much time it took to handle the stimulus and the result or condition of the entity after the stimulus.

In simulations involving several processes within an entity, stimulus reaction is often shown at an individual process level along with the stimulus transfer between processes and the overall effect of the stimulus.

Depending on the complexity of a simulated entity, a particular stimulus might undergo a series of alterations between processes that could result in a change in the sequence or number of processes affected by the stimulus.

To begin any simulation program, you must carefully design and define every component involved in the operation of the object being simulated and their various attributes.

Next, you must define each activity to be considered in the simulation. Each activity must be further defined into the event or events (processes) which make up the functioning of a particular activity.

Each event requires a set of attributes defining the statistics to be collected about the event and the information concerning the components involved in the event.

Furthermore, the events must have a sequence value associated with them in order to configure the priority of operation of the events in an activity. Similarly, activities must be sequenced by order of priority in their occurance with each other.

In many simulations, some of the components involved in the simulation are not directly part of the entity being simulated. These components are the stimuli interacting with the entity. Their occurances within the progress of a simulation are usually based on either a random function of time and number or by a particular proportionate distribution of time and number such as a mean or logarithum distribution.

In simulations where all the components involved are part of the entity being simulated, the stimulus to make the simulation work is usually time itself. In these cases, the simulation model is usually run for a specific period of time in order to determine how much work the entity can produce in that period of time.

In all cases, simulations are based on time in that all events that occur during the progress of a simulation are tracked by time of occurance.

Now that we have the necessary ground rules for simulation program design, let us take a look at an example of a simulation model. In this example, we will simulate an elevator system in an office building. First, let us take a look at the components involved in the elevator system entity: first, we have the building itself.

This component is important in that it supplies us with some much needed information for our simulation via its attributes. These attributes are the number of floors in the building and the distance between floors that the elevator cars must travel.

This gives rise to our next component in the simulation, the floors. Each floor is important as a unique component in the simulation because they keep track of the number of people on a particular floor. These people will undergo a random selection process in the simulation which will select those people who will change floors.

Our major component in the simulation is the elevator cars themselves. Each elevator car must maintain some specific attributes during the course of our simulation. The attributes are passenger capacity, current passenger load, car travel speed, minimum car starting/stopping and door opening/closing times, and current floor location. Of course, we also need to know the number of elevator cars in the building.

Lastly, we need to introduce a component that will stimulate activity in our model. This component is people. They will randomly arrive at the building, choose a floor, wait on a floor, change floors, and exit the building.

Additionally, the time taken to walk to and from an elevator, walk into and out of the building, and walk into and out of an elevator car must be taken into account during certain activities in the simulation.

Now let us see how we can represent all of our components and related information we need in our simulation through the use of BASIC instructions: we must make use of some Level II BASIC instructions to accomplish this.

Since we will need to use several tables of related information, we can use the "DIM" instruction to define more than one table with names of our choosing. These tables and other related fields must be initialized with starting values. Many can be assigned by the use of the input "literal" instruction, where "literal" acts as a prompt asking for the field or table element value to be inputted.

Here is an example of an elevator simulation initialization routine:

```
10 INPUT "NUMBER OF FLOORS"; N1
20 INPUT "DISTANCE BETWEEN FLOORS IN FEET"; N2
30 INPUT "NUMBER OF ELEVATORS"; N3
40 INPUT "WALKING TIME IN OR OUT OF BUILDING IN SECONDS"; N4
50 INPUT "WALKING TIME TO OR FROM AN ELEVATOR IN SECONDS"; N5
60 INPUT "WALKING TIME IN OR OUT OF AN ELEVATOR IN SECONDS"; N6
70 REM DEFINE AND INITIALIZE TO Ø THE NUMBER OF PEOPLE WAITING FOR AN ELEVATOR
   ON EACH FLOOR FOR EACH ELEVATOR
80 DIM A1(N1)
90 FOR I=1 TO N1
95 FOR J=1 TO N3
100 A1(I,J)=Ø
105 NEXT J
```

```
110 NEXT I
120 REM DEFINE AND INITIALIZE ELEVATORS
130 DIM A2(N3,7)
140 FOR I=1 TO N3
150 INPUT "PASSENGER CAPACITY"; A2(I,1)
160 REM INITIALIZE PASSENGER LOAD TO Ø
170 A2(I,2)=Ø
180 INPUT "CAR TRAVEL SPEED IN FEET/SECOND"; A2(I,3)
190 INPUT "CAR STARTING OR STOPPING TIME IN SECONDS"; A2(I,4)
200 INPUT "DOOR OPENING OR CLOSINE TIME IN SECONDS"; A2(I,5)
210 REM INITIALIZE CURRENT FLOOR LOCATION TO 1 AND DIRECTION TO UP
215 A2(I,7)=1
220 A2(I,6)=1
225 REM IN A2(I,7), 1=UP, 2=DOWN IN DIRECTION
230 NEXT I
240 REM INITIALIZE SIMULATION TIME CLOCK AND TOTAL NUMBER OF PEOPLE IN THE
    BUILDING TO Ø
250 N7=Ø
260 N9=Ø
```

Note that the elevator table A2 is a two dimensional table containing six attribute fields for each elevator in the building. The simulation time clock will advance as each activity scheduled through the simulation takes place.

Now let us take a look at the activities and their associated events involved in our simulation. These activities include:

1. A person enters the building.
2. A person exits the building.
3. Elevator discharges passengers.
4. Elevator loads passengers.
5. Elevator travels to another floor.
6. A person walks to the elevator.
7. A person walks from the elevator.

Since none of the above activities involves a series of complex processes, our simulation does not need to sub-define any activities into a group of interlocking events.

As an example of how you might view the sub-division of an activity into a group of events, here is how some of our activities might look:

The "elevator travels to another floor" activity would need:

A) Shut elevator door on present floor.
B) Travel distance to desired floor.
C) Open elevator door on new floor.

In order for the elevator system to send cars to the proper floors, each elevator car must keep track of the number of passengers bound for each of the possible floors in the building. This informational table must be initialized to 0 for the beginning of the simulation.

This can be done by the following BASIC instructions:

```
300 DIM A3(N3,N1)
310 FOR I=1 TO N3
320 FOR J=1 TO N1
330 A3(I,J)=Ø
340 NEXT J
350 NEXT I
```

As each activity is required within the simulation, it must be scheduled for completion by the activity whose completion has initiated that activity. This schedule of activities must be kept in a table organized by completion time sequence. This table must also be initialized before the beginning of the simulation.

This could be done with the following BASIC instructions:

```
400 N8=(N3 * N1 * 7)+1ØØ
410 DIM A4(N8,5)
420 FOR I=1 TO N8
430 FOR J=1 TO 5
440 A4(I,J)=Ø
450 NEXT J
460 NEXT I
```

As you can see from the above code, the maximum number of scheduled activities in our simulation as stored in field N8 is equal to an allowance for the number of possible people walking to an elevator plus the number of elevators times the number of floors times the number of different activities (7) - this can be changed.

Table field A4(I,1) holds the scheduled completion time of the activity. Table field A4(I,2) holds the identifying number of the activity. Table field A4(I,3) holds the identifying number of any elevator involved in the activity. Table field A4(I,4) holds the identifying number of the floor involved in the activity. Table field A4(I,5) holds the number of people to be affected by the completion of the scheduled activity.

Now we need a group of seven BASIC subroutines, each of which will duplicate the actions that must occur upon the completion of one of the seven activities required within our simulation.

The "person enters the building" activity must add a person to our simulation model and must schedule him for a "walk to the elevator" activity.

To do this, we will also need a BASIC subroutine which will store an activity to be scheduled for completion in the activity schedule table. This subroutine involves searching the existing activity table for a place to store our scheduled activity.

This empty position is recognizible by a zero in the table's A4(I,2) activity number field. If no empty position can be found, the simulation must be stopped and the table size increased through additional instructions in our activity table initialization routine.

The BASIC instruction code to store a scheduled activity in the table is as follows:

```
1000 FOR I=1 TO N8
1010 IF A4(I,2)=Ø THEN 1Ø5Ø
1020 NEXT I
1030 PRINT "SCHEDULED ACTIVITY TABLE FULL!"
1040 STOP
1050 A4(I,1)=P1
1060 A4(I,2)=P2
1070 A4(I,3)=P3
1080 A4(I,4)=P4
1090 A4(I,5)=P5
1095 RETURN
```

Note the use of the BASIC command "RETURN". When this subroutine is activated by a "GOSUB 1000" BASIC instruction anywhere in our simulation program, the section of the program containing this instruction will pause and allow the activity table store subroutine to be executed. The RETURN instruction will then resume execution of the simulation program at the instruction immediately following the GOSUB in the program section paused.

Before executing a GOSUB 1000, we must fill the P1, P2, P3, P4 and P5 fields with the values associated with the activity to be scheduled and stored in the table.

The P1 field must contain the time that the activity will be completed. The P2 field must contain the identifying number of the activity being scheduled. The P3 field must contain the identifying number of the elevator used in the activity, if any. The P4 field must contain the identifying number of the floor involved in the activity, if any. The P5 field must contain the number of people affected by the completion of the activity, if any.

Now we can take a look at the BASIC routine that will handle the completion of a person entering the building: note that the elevator selected to walk to is chosen at random from the number of elevators in our model.

```
2000 P1=N7+N4+N5
2010 RANDOM
2020 P3=RND(N3)
2030 P2=6
2040 P4=1
2050 P5=1
2060 GOSUB 1ØØØ
2070 REM ADD ONE TO THE TOTAL NUMBER OF PEOPLE IN THE BUILDING
2080 N9=N9+1
2090 RETURN
```

When a person exits the building, no further activity can be scheduled for that person. The only thing the simulation needs to do is deduct that person from the total number of people in the building. If all the people have left the building, the simulation ends.

Here are the BASIC instructions needed to handle this type of activity:

```
3000 N9=N9-1
3010 IF N9.GT.0 THEN 3040
3020 PRINT "ALL THE PEOPLE HAVE LEFT THE BUILDING!"
3030 END
3040 RETURN
```

Once a person in our simulation has walked to an elevator, he must be scheduled to load into an elevator. This involves indicating that person's wait on his present floor by adding one to the total number of people waiting on that floor.

Here are the BASIC instructions needed to do this:

```
4000 A1(P4,P3)=A1(P4,P3)+1
4010 REM NOTE THAT THE P3 AND P4 FIELDS RETAIN THE ELEVATOR AND FLOOR NUMBERS
     FROM THE WALK ACTIVITY
4020 RETURN
```

As elevators move from floor to floor, they are scheduled to discharge and load passengers as their activities. Since, in our simulation model, all of our elevator cars start at the first floor, we must schedule an initial passenger load activity for all elevators on the first floor at the start of our simulation.

From that point on, the passenger load and discharge activities as well as elevator car movement (a non-scheduled, transparent activity occuring between passenger loads and discharges) are scheduled automatically as elevators are requested on different floors.

If an elevator is empty and not required on any floor, it will automatically be scheduled to wait for passenger load on the first floor.

Here are the initial passenger load BASIC instructions required at simulation start-up:

```
500 FOR I=1 TO N3
505 REM SET TIME FOR DOOR HANDLING
510 P1=N7+N6+N4+N5+(A2(I,5)*2)
520 P2=4
530 P3=I
540 P4=1
550 P5=Ø
560 GOSUB 1ØØØ
570 NEXT I
```

When a passenger load activity is completed, the elevator's next movement, passenger discharge (if any) and next passenger load must be scheduled. If there are passengers taken on or remaining in the elevator car from past loads, a passenger discharge activity must be scheduled. If, at the destination floor, more passengers are waiting, additional passenger loads and discharges will be scheduled.

Here are the BASIC instructions needed to handle the completion of a passenger

load activity:

```
5000 J=A1(P4,P3):A2(P3,6)=P4:A2(P3,2)=A2(P3,2)+J
5010 A1(P4,P3)=Ø:IF A2(P3,2).GT.A2(P3,1) THEN J=J-(A2(P3,2)-A2(P3,1)):A1(P4,P3)=
     A2(P3,2)-A2(P3,1):A2(P3,2)=A2(P3,1)
5015 IF J=Ø THEN 5Ø75
5020 FOR I=1 TO J
5025 REM DETERMINE DESTINATION FLOORS
5030 RANDOM
5040 K=RND(N1)
5050 IF K=P4 THEN 5Ø4Ø
5060 A3(P3,K)=A3(P3,K)+1
5070 NEXT I
5075 K=Ø
5080 IF A2(P3,7)=1 THEN P4=P4+1 ELSE P4=P4-1
5090 IF P4.GT.N1 THEN A2(P3,7)=2:P4=A2(P3,6)-1
5100 IF P4=Ø THEN A2(P3,7)=1:P4=A2(P3,6)+1
5110 K=K+1
5120 IF K=N1 THEN P4=1:A2(P3,7)=1:GOTO 5160
5130 IF A3(P3,P4).GT.Ø THEN 5160
5140 IF A1(P4,P3).GT.Ø AND A2(P3,2).LT.A2(P3,1) THEN 5160
5150 GOTO 5080
5160 IF A3(P3,P4).GT.Ø THEN 5300
5165 M=A1(P4,P3):IF A2(P3,2)+A1(P4,P3).GT.A2(P3,1) THEN M=M-((A2(P3,2)+A1(P4,P3)
     )-A2(P3,1))
5170 P1=(M*N6)+N7+((ABS(A2(P3,6)-P4)*N2)/A2(P3,3))+A2(P3,4)+A2(P3,5)
5180 P2=4
5190 P5=M
5200 GOSUB 1ØØØ
5210 RETURN
5300 P1=(A3(P3,P4)*N6)+N7+((ABS(A2(P3,6)-P4)*N2)/A2(P3,3))+A2(P3,4)+A2(P3,5)
5310 P2=3
5320 P5=A3(P3,P4)
5330 GOSUB 1ØØØ
5340 RETURN
```

Note that in the above subroutine, we must first determine how many passengers waiting to be loaded can actually fit in our elevator car and then adjust the passenger pick up value in field J accordingly.

Next we must randomly select a floor as the destination of each passenger taken aboard and update the passenger discharge table accordingly.

Then we must determine the next closest floor in our elevator's direction of travel that has either been selected by people in the elevator car as their destination floor, or contains people waiting to be loaded into the elevator car, providing there is room in the elevator car for at least a portion of those people. If no such floor can be found in our direction of travel, the direction of travel is reversed and the search continues.

Once we have examined each floor in the building for both of the above conditions and found none to exist, the elevator is scheduled to wait at the first

floor.

If a floor requiring passenger discharge or load is found, the corresponding activity is scheduled for completion.

Note that the time of completion is computed by the sum of the total elevator entry or exit time for the passengers involved plus the total travel time between floors, if any floor change is required to complete the activity, plus the elevator motor and door operation times.

Because of the elevator car's maximum capacity, the loading of passengers may result in only a portion of the people waiting for the elevator on a floor to be loaded with the remainder of the people rescheduled for passenger loading when the elevator stops on that floor again.

Also, if the next floor that the elevator will travel to is scheduled for both passenger loading and discharging, the passenger discharge activity will be completed first so that the elevator car's capacity is at its highest for the passenger loading.

Once the passenger discharge activity is completed, we must compute when the passengers will be returning to the elevator. This time must include the walking time to and from the elevator plus a random amount of time for their stay on that floor. Again, the "walk from the elevator" activity is transparent within these processes.

If the passengers are being discharged on the first floor, they will be scheduled to leave the building rather than returning to the elevator for further floor changes.

Once all the passengers who are discharging have been scheduled for building exit or a return to the elevator, the elevator must be scheduled for its next loading or discharging activity. This can easily be done by re-using the portion of the passenger loading completion subroutine that accomplishes that task.

Here are the BASIC instructions which will handle the passenger discharge activity completion as outlined above:

```
6000 J=A3(P3,P4):A3(P3,P4)=Ø:A2(P3,6)=P4
6005 A2(P3,2)=A2(P3,2)-J
6010 FOR I=1 TO J:RANDOM:IF P4=1 THEN 6Ø9Ø
6020 P1=N7+(N5*2)+RND(300)
6030 REM NOTE THE RANDOM STAY OF A PERSON ON A FLOOR BEFORE
6040 REM RETURNING TO THE ELEVATOR HAS A MAXIMUM OF 5 MINUTES
6050 REM OR 300 SECONDS - THIS MAY BE CHANGED
6060 P2=6
6070 P5=1
6080 GOTO 6120
6090 P1=N7+N5+N4
6100 P2=2
6110 P5=1
6120 GOSUB 1ØØØ
6130 NEXT I
6135 REM IF PEOPLE ARE WAITING ON THIS FLOOR, LOAD THEM FIRST ELSE SEARCH FOR
     NEXT ACTIVITY
6140 IF A1(P4,P3).GT.Ø THEN 5165 ELSE 5075
```

Now that we have all the routines needed to handle the various activities in our simulation model, we need a central routine to coordinate the running of the simulation. The main requirements of this last routine are in two parts: the releasing of the completed activities from the scheduled activity table and the reporting of statistics collected by the simulation about its activities and their resulting effects.

To release completed activities, we will search the activity table for the lowest completion time value - the next activity to be completed, set the simulation time clock to that time, and GOSUB the appropriate subroutine for that activity's completion.

To report statistics, the program can display each activity as it is completed. Additionally, whenever the keyboard interrupts the simulation through the user hitting any key on the keyboard after the simulation has started completing activities, a report of the current status of our simulation model will be displayed.

The reporting structure used in our simulation example can be changed into any form desired in any simulation you might write. Our elevator simulation model is merely an example for you to follow, as to technique, in your simulation programming requirements.

Here are the BASIC instructions needed to handle the above functions:

```
600 A$="  ":A$=INKEY$:IF A$="  " OR A$="" THEN 720
605 REM KEYBOARD INTERRUPT ACCEPTED
610 FOR I=1 TO N3:PRINT "ELEVATOR";I;"CURRENT FLOOR";A2(I,6);"DIRECTION";A2(I,7);
    "CAPACITY";A2(I,2)
620 PRINT "PEOPLE WAITING ON FLOORS"
630 FOR J=1 TO N1
640 PRINT J;"-";A1(J,I);
650 NEXT J:PRINT" "
660 PRINT "PEOPLE TRAVELING TO FLOORS"
670 FOR J=1 TO N1
680 PRINT J;"-";A3(I,J);
690 NEXT J:PRINT" "
695 INPUT "HIT ENTER TO CONTINUE";A$
700 REM THE ABOVE PRINTS THE FLOOR NUMBER - NUMBER OF PEOPLE
705 NEXT I
710 PRINT N9;"PEOPLE IN BUILDING";"TIME";N7
720 K=99999:J=Ø
730 FOR I=1 TO N8:IF A4(I,1)=Ø THEN 75Ø
740 IF A4(I,1).LT.K THEN J=I:K=A4(I,1)
750 NEXT I
760 P1=A4(J,1):P2=A4(J,2):P3=A4(J,3):P4=A4(J,4):P5=A4(J,5)
765 FOR M=1 TO 5:A4(J,M)=Ø:NEXT M
770 PRINT "TIME;P1;"ACTIVITY";P2;"ELEVATOR";P3;"FLOOR";P4;"PEOPLE";P5
780 K=P2:N7=P1
785 REM CALL THE PROPER ACTIVITY SUBROUTINE ACCORDING TO ACTIVITY NUMBER
790 ON K GOSUB 2ØØØ, 3ØØØ, 6ØØØ, 5ØØØ, 7ØØØ, 4ØØØ, 7ØØØ
800 GOTO 6ØØ
```

```
7000 RETURN
7005 REM DUMMY ACTIVITY RETURN - TRANSPARENT ACTIVITY NUMBERS USE IT
```

In conclusion, this article and its elevator simulation example should give you an idea of how a simulation works. This technique can be expanded to handle any number of activities and table A4 can be expanded to hold a scheduled event code for activities requiring multiple events. The activity subroutines would then use "ON GOSUB" instructions to execute associated event subroutines.

Also, individual activity subroutines could perform separate statistics collection and reporting using additional tables.

The above elevator system example can be expanded and modified in many ways as well - but we will leave that to your imagination!

....

THE 80 NOTEBOOK
R.D.#3
Nazareth, PA 18064

BULK RATE
U.S. POSTAGE
PAID
Stockertown, PA 18083
Permit No. 8